Investigation into Gaussian Quadrature

Jack Deye
email@g.ucla.edu

Zachary Diamond
zacharydiamond@g.ucla.edu

Jonathan Levi
email@g.ucla.edu

Reeshad Mohammed
email@g.ucla.edu

March 11, 2025

Contents

1 Introduction

Quadrature is the name given to various methods used to approximate integrals. Some integrals are impossible or unfeasible to compute analytically (such as $\int e^{x^2}$), which necessitates the need to approximate them numerically. Thus, various methods have been developed to approximate these integrals, such as the Trapezoid Rule, Simpson's Rule, and Gaussian Quadrature. The aforementioned quadratures have varying levels of necessary computation and error. We intend to investigate and compare these quadrature forms. In particular, we will analyze the accuracy of Gaussian Quadrature to the more simple Trapezoid Rule and Simpson's Rule. Additionally, we will compare the theory of Guassian Quadrature convergence to experimental convergence.

2 Quadrature Techniques

2.1 Trapezoid Rule

The Trapezoid Rule is developed using Lagrange polynomials and equally spaced nodes. Consider $\int_a^b f(x)dx$. When we replace $f(x)$ with the first Lagrange polynomial approximation, with $x_0 = a$ and $x_1 = b$, and integrate, we get

$$\frac{(x_1 - x_0)}{2}[f(x_0) + f(x_1)] - \frac{(x_1 - x_0)^3}{12}f''(\xi)$$

Where $\xi \in (x_0, x_1)$. Naturally, we can set $h = x_1 - x_0$, resulting in the Trapezoid Rule:

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}[f(x_0) + f(x_1)] - \frac{h^3}{12}f''(\xi)$$

It is clear that this approach only works well on quite small intervals; so, we develop the Composite Trapezoid Rule. The Composite Trapezoid Rule applies the Trapezoid Rule on n subintervals within the initial interval. The formula is as follows:

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{2}[f(x_0) + 2\sum_{j=1}^{n-1} f(x_j) + f(x_n)] - \frac{x_n - x_0}{12}h^2 f''(\mu)$$

Where $\mu \in (x_0, x_n)$.

2.2 Simpson's Rule

Similar to the Trapezoid Rule, Simpson's Rule is also developed using Lagrange polynomials. Simpson's Rule differs in that it uses the second Lagrange polynomial. Consider $\int_a^b f(x)dx$. We set $x_0 = a$, $x_1 = a + h$, and $x_2 = b$, where $h = \frac{b-a}{2}$. Now, when we replace $f(x)$ with the second Lagrange polynomial approximation and integrate, we get Simpson's Rule:

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90}f^{(4)}(\xi)$$

Where $\xi \in (x_0, x_2)$. However, like the Trapezoid Rule, this form only works well on small intervals. The Composite Simpson's Rule is developed similarly to the Composite

Trapezoid Rule, splitting the interval into n subintervals, where n is an even integer. The formula is as follows:

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{3}[f(x_0) + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(x_{2j-1}) + f(x_n)] - \frac{x_n - x_0}{180}h^4 f^{(4)}(\mu)$$

Where $\mu \in (x_0, x_n)$.

2.3 Guassian Quadrature

Gaussian quadrature is particularly efficient for integrating polynomials and uses specially chosen points and weights to achieve high accuracy.

Gaussian quadrature approximates integrals using:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where x_i are the nodes and w_i are the weights. The nodes and weights are chosen such that the method is exact for polynomials of degree $2n-1$ or lower. Typically, this integral is from $[-1, 1]$.

2.3.1 Motivation

Consider the trapezoid rule, which only provide exact solutions to integrals for linear functions. But we can do better, take n nodes $x_1, x_2, \cdots, x_n$. We can improve the trapezoid by choosing the weights such that they are exact for linear, quadratic, cubic, up to polynomials of degree $n-1$.

$$\int_{-1}^{1} f \approx w_1 f(x_1) + w_2 f(x_2) + \cdots + w_n f(x_n) \tag{1}$$

$$f(x) = 1 \to \int_{-1}^{1} 1\,dx = 2 = w_1 + w_2 + \cdot + w_n$$

$$f(x) = x \to \int_{-1}^{1} x\,dx = 0 = w_1 x_1 + w_2 x_2 + \cdot + w_n x_n$$

$$\vdots$$

$$f(x) = x^{n-1} \to \int_{-1}^{1} x^{n-1}\,dx = 0 = w_1 x_1^{n-1} + w_2 x_2^{n-1} + \cdots + w_n x_n^{n-1}$$

Resulting in the following linear equation (with a Vandermonde Matrix):

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & \ddots & \cdots & x_n \\ \vdots & & \ddots & \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ \vdots \\ b_i \end{bmatrix}$$

In practice, this method is not feasible because Vandermonde matrices are badly ill-conditioned, as the condition number increases exponentially, (https://arxiv.org/abs/1504.02118), resulting in slight changes in b causing large changes in w_i's.

2.3.2 Derivation

We develop the formula for Gaussian quadrature using Legendre Polynomials - a series of orthogonal polynomials obtained by performing the Gram-Schmidt process on the standard basis for polynomials of degree n and the inner product $\langle p, q\rangle = \int_{-1}^{1} p(x)q(x)\,dx$. The first few Legendre Polynomials are as follows:

$$\begin{aligned} L_0(x) =& 1 \\ L_1(x) =& x \\ L_2(x) =& \frac{1}{2}(3x^2 - 1) \\ &\vdots \end{aligned}$$

Note that $L_i(x)$ is orthogonal to all polynomials of degree less than i, meaning $\forall p \in \mathbb{P}$, $\langle L_i, p\rangle$ for $\deg(p) \leq i$. Importantly, they also have exactly n roots over $\mathbb{R}$.

Now, given a polynomial $p(x)$ of degree $2n-1$, we can divide it by L_n, resulting in the following division with degrees:

$$\overbrace{p(x)}^{2n-1} = \overbrace{q(x)}^{n-1}\overbrace{L_n(x)}^{n} + \overbrace{r(x)}^{n-1} \tag{2}$$

Integrating both sides, we get

$$\int_{-1}^{1} p(x)\,dx = \int_{-1}^{1} q(x)L_n(x)\,dx + \int_{-1}^{1} r(x)\,dx = 0 + \int_{-1}^{1} r(x)\,dx$$

because the orthogonality of $L_n(x)$.
Going back to quadrature, we can ensure the same behavior by picking nodes at the zeros of $L_n(x)$.
Because $\int_{-1}^{1} p(x)\,dx = \int_{-1}^{1} r(x)\,dx$, we can interpolate $r(x)$ exactly with Lagrange polynomials, resulting in

$$\int_{-1}^{1} r(x)\,dx = \int_{-1}^{1} \left(\sum_{i=1}^{n} f(x_i)l_i(x)\right)\,dx = \sum_{i=1}^{n} f(x_i)\int_{-1}^{1} l_i(x)\,dx$$

meaning we choose our weights the be the integral of the Lagrange basis polynomials, or

$$w_i = \int_{-1}^{1} \prod_{j=1 j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\,dx$$

This results in perfect interpolation of polynomials of degree $2n-1$, with n nodes. We can relate Lagrange and Legendre polynomials and then use the Christoffel-Darboux formula to rewrite the definition of the weights as

$$w_i = \frac{2}{(1 - x_i^2)(L_n'(x_i))^2} \tag{3}$$

Before using Gaussian Quadrature on any function, one must change an integral over $[a, b]$ to one over $[-1, 1]$. This can be done with

$$\int_{a}^{b} f(x)\,dx = \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)\frac{b-a}{2}\,dx$$

Resulting in a formula of:

$$\int_a^b f(x) \approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{b-a}{2} x_i + \frac{a+b}{2}\right) \tag{4}$$

3 Comparison of Quadrature

3.1 Composite Trapezoid Rule MATLAB Code

```
function I = composite_trapezoid_rule(f, a, b, n)
    % COMPOSITE_TRAPEZOID_RULE Calculates integral approximation based off
    % the composite trapezoid rule, given function handle f, lower bound a,
    % upper bound b, subintervals n
h = (b - a) / n;
x = a:h:b;
I = h / 2 * (f(a) + 2*sum(f(x(2:end-1))) + f(b));
end
```

3.2 Composite Simpson's Rule MATLAB Code

```
function I = composite_simpsons_rule(f, a, b, n)
    % COMPOSITE_SIMPSONS_RULE Calculates integral approximation based off
    % the composite simpson's rule, given function handle f, lower bound a,
    % upper bound b, subintervals n (must be even)
if mod(n, 2) ~= 0
    error("n Must be even");
else
    h = (b - a) / n;
    x = a:h:b;

    odds = 2:2:n;
    evens = 3:2:n-1;

    I = h / 3 * (f(x(1)) + f(x(n+1)) + 4 * sum(f(x(odds))) + 2 * sum(f(x(ev
end
```

3.3 Gauss-Legendre Quadrature MATLAB Code

```
function I = gauss_legendre_quadrature(f, a, b, n)
    % f is a function handle: such as @(x) sin(x).^2
    % Can also use naive_nodes_weights, but it is much slower
 [nodes, weights] = GW_nodes_weights(n);
 transformed_nodes = ((b - a) / 2) * nodes + (a + b) / 2;
 I = ((b - a) / 2) * sum(weights .* f(transformed_nodes));
end

function [nodes, weights] = GW_nodes_weights(n)
   % Golub-Welsch algorithm
   % GLINT  Gauss-Legendre numerical integration.
```

```
    % Fundamentals of Numerical Computation, By Tobin A. Driscoll and Richar
    % https://github.com/tobydriscoll/fnc-extras/blob/master/fnc/glint.m
     beta = 0.5 ./ sqrt(1-(2*(1:n-1)).^(-2));
     T = diag(beta, 1) + diag(beta, -1);
     [V, D] = eig(T);
     nodes = diag(D);
     weights = 2 * (V(1, :).^2)';
     [nodes, idx] = sort(nodes);
     weights = weights(idx);
end
```

4 Order of Convergence

5 Chebyshev-Gauss Weights

6 Conclusion

7 References

1. https://www.math.umd.edu/ mariakc/AMSC466/LectureNotes/quadrature.pdf, page 22 error estimates
2. Numerical Analysis, Richard L. Burden, J. Douglas Faires, etc.